**Assignment 9.3**

**Nosql Databases**

* NoSQL Databases (DBs) are called Non-Relational or Distributed Databases.
* Data Storage is not based on a single data model. Most outstanding ones are key-value pair, graph, document, and columnar.
* NoSQL DBs use UnQL (Unstructured Query Language) for querying. The syntax of using UnQL varies from database to database.
* Schemas are dynamic. Each row need not have data for each column.
* Database scaling is possible with horizontal scaling across multiple servers.
* NoSQL database follows the Brewers CAP theorem (Consistency, Availability and Partition tolerance).
* NoSQL DBs get classified based on the data storage type- Graph DBs, Key-Value store DBs, and Document store DBs.

**Types of Nosql Databases:**

MongoDB :

* It is non-relational database with dynamic schema.
* It is written in C++ language.
* It stores data in JSON like documents.
* Querying is done using a single document which eliminates the join operations.
* Document databases pair each key with a complex data structure known as a document.
* Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.

Couch DB:

* CouchDB is an acronym for Cluster Of Unreliable Commodity Hardware & is a document based NoSQL database.
* A document-oriented database designed for Web applications.
* JavaScript is its query language, using MapReduce-and HTTP for an API – using your web browser.
* Scaling is done with horizontal partitioning of database.

Cassandra and HBase:

* Wide-column stores such as Cassandra and HBase are optimized for queries over large datasets, and store columns of data together, instead of rows.
* Schemas are dynamic. Each row need not have data for each column.
* Key-value stores are the simplest NoSQL databases
* Every single item in the database is stored as an attribute name (or 'key'), together with its value.

Graph stores are used to store information about networks of data, such as social connections. Graph stores include Neo4J and Giraph.

**CAP Theorem:**

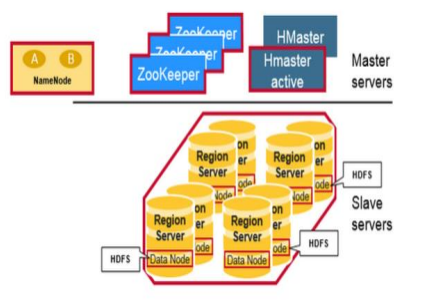
* **Consistency** - This means that the data in the database remains consistent after the execution of an operation. For example after an update operation, all clients see the same data.
* **Availability** - This means that the system is always on (service guarantee availability), no downtime.
* **Partition Tolerance** - This means that the system continues to function even if the communication among the servers is unreliable, i.e. the servers may be partitioned into multiple groups that cannot communicate with one another.
* Duplicate Copy of same data is maintained on Multiple Machines.
* This increases availability, but decreases consistency.
* If data on one machine changes, the update propagates to the other
* Machine, system is inconsistent, but will become eventually consistent.
* If duplicate copy of same data is not maintained, consistency is superior but availability decreases.
* Machine, system is inconsistent, but will become eventually consistent.

**HBASE Architecture**

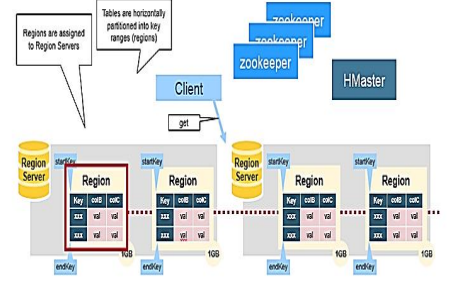
Physically, HBase is composed of three types of servers in a master slave type of architecture. Region servers serve data for reads and writes. When accessing data, clients communicate with HBase RegionServers directly. Region assignment, DDL (create, delete tables) operations are handled by the HBase Master process. Zookeeper, which is part of HDFS, maintains a live cluster state.

The Hadoop DataNode stores the data that the Region Server is managing. All HBase data is stored in HDFS files. Region Servers are collocated with the HDFS DataNodes, which enable data locality (putting the data close to where it is needed) for the data served by the RegionServers. HBase data is local when it is written, but when a region is moved, it is not local until compaction.

The NameNode maintains metadata information for all the physical data blocks that comprise the files.



HBase Tables are divided horizontally by row key range into “Regions.” A region contains all rows in the table between the region’s start key and end key. Regions are assigned to the nodes in the cluster, called “Region Servers,” and these serve data for reads and writes. A region server can serve about 1,000 regions.



Region assignment, DDL (create, delete tables) operations are handled by the HBase Master.

A master is responsible for:

* Coordinating the region servers

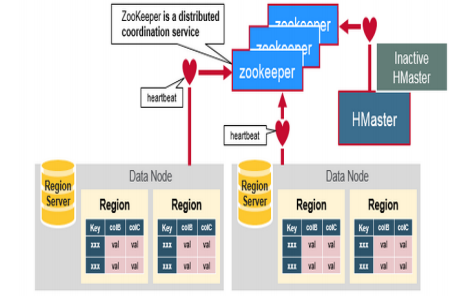
- Assigning regions on startup , re-assigning regions for recovery or load balancing

- Monitoring all RegionServer instances in the cluster (listens for notifications from zookeeper)

* Admin functions

- Interface for creating, deleting, updating tables

HBase uses ZooKeeper as a distributed coordination service to maintain server state in the cluster. Zookeeper maintains which servers are alive and available, and provides server failure notification. Zookeeper uses consensus to guarantee common shared state. Note that there should be three or five machines for consensus.



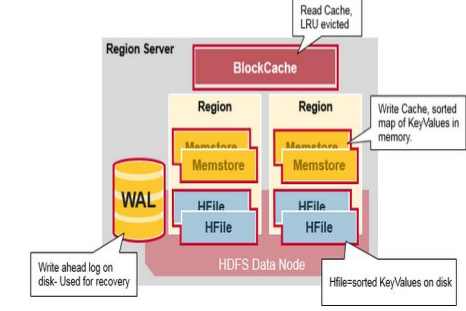
*There is a special HBase Catalog table called the META table, which holds the location of the regions in the cluster. ZooKeeper stores the location of the META table.*

This is what happens the first time a client reads or writes to HBase:

1. The client gets the Region server that hosts the META table from ZooKeeper.
2. The client will query the .META. server to get the region server corresponding to the row key it wants to access. The client caches this information along with the META table location.
3. It will get the Row from the corresponding Region Server.

For future reads, the client uses the cache to retrieve the META location and previously read row keys. Over time, it does not need to query the META table, unless there is a miss because a region has moved; then it will re-query and update the cache.

A Region Server runs on an HDFS data node and has the following components:

* **WAL**: Write Ahead Log is a file on the distributed file system. The WAL is used to store new data that hasn't yet been persisted to permanent storage; it is used for recovery in the case of failure.
* **BlockCache**: is the read cache. It stores frequently read data in memory. Least Recently Used data is evicted when full.
* **MemStore**: is the write cache. It stores new data which has not yet been written to disk. It is sorted before writing to disk. There is one MemStore per column family per region.
* **Hfiles** store the rows as sorted KeyValues on disk.
* 

When the client issues a Put request, the first step is to write the data to the write-ahead log, the WAL:

* - Edits are appended to the end of the WAL file that is stored on disk.
* - The WAL is used to recover not-yet-persisted data in case a server crashes.

Once the data is written to the WAL, it is placed in the MemStore. Then, the put request acknowledgement returns to the client.

The MemStore stores updates in memory as sorted KeyValues, the same as it would be stored in an HFile. There is one MemStore per column family. The updates are sorted per column family.

When the MemStore accumulates enough data, the entire sorted set is written to a new HFile in HDFS. HBase uses multiple HFiles per column family, which contain the actual cells, or KeyValue instances. These files are created over time as KeyValue edits sorted in the MemStores are flushed as files to disk.

Note that this is one reason why there is a limit to the number of column families in HBase. *There is one MemStore per CF*; when one is full, they all flush. It also saves the last written sequence number so the system knows what was persisted so far.

The highest sequence number is stored as a meta field in each HFile, to reflect where persisting has ended and where to continue. On region startup, the sequence number is read, and the highest is used as the sequence number for new edits.

*Data is stored in an HFile which contains sorted key/values*. When the MemStore accumulates enough data, the entire sorted KeyValue set is written to a new HFile in HDFS. This is a sequential write. It is very fast, as it avoids moving the disk drive head.

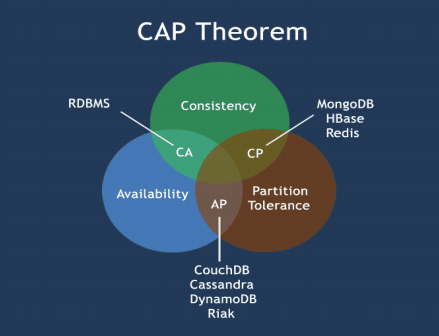
An HFile contains a multi-layered index which allows HBase to seek to the data without having to read the whole file. The multi-level index is like a b+tree:

* Key value pairs are stored in increasing order
* Indexes point by row key to the key value data in 64KB “blocks”
* Each block has its own leaf-index
* The last key of each block is put in the intermediate index
* The root index points to the intermediate index

The trailer points to the meta blocks, and is written at the end of persisting the data to the file. The trailer also has information like bloom filters and time range info. Bloom filters help to skip files that do not contain a certain row key. The time range info is useful for skipping the file if it is not in the time range the read is looking for.

We have seen that the KeyValue cells corresponding to one row can be in multiple places, row cells already persisted are in Hfiles, recently updated cells are in the MemStore, and recently read cells are in the Block cache. So when you read a row, how does the system get the corresponding cells to return? A Read merges Key Values from the block cache, MemStore, and HFiles in the following steps:

1. First, the scanner looks for the Row cells in the Block cache - the read cache. Recently Read Key Values are cached here, and Least Recently Used are evicted when memory is needed.
2. Next, the scanner looks in the MemStore, the write cache in memory containing the most recent writes.
3. If the scanner does not find all of the row cells in the MemStore and Block Cache, then HBase will use the Block Cache indexes and bloom filters to load HFiles into memory, which may contain the target row cells.



**HBase vs RDBMS**

|  |  |
| --- | --- |
| RDBMS | NOSQL |
| RDBMS is row-oriented databases | HBase is a distributed, column- oriented data storage system |
| RDBMS tables have fixed-schema | Hbase tables do not have fixed- schema |
| RDBMS tables guarantee ACID  Properties | Hbase tables guarantee consistency and partition tolerance |
| RDBMS uses SQL (Structured query Langauge ) to query the data | Hbase uses Java client API |
| Data Storage is based on a single data model – relational model. This model is table based i.e. Consisting of rows and columns of data. | Data Storage is not based on a single data model. Most outstanding ones are key-value pair, graph, document, and columnar. |
| SQL Databases are called Relational databases or RDMS – Relational Database Management Systems. | NoSQL Databases (DBs) are called Non-Relational or Distributed Databases. |
| SQL DBs use only SQL as the language for querying. The syntax of SQL is standard across any database. | NoSQL DBs use UnQL (Unstructured Query Language) for querying. The syntax of using UnQL varies from database to database. |
| Data stored as per fixed schemas. Each row must have data specific to a column. | Schemas are dynamic. Each row need not have data for each column. |
| Database scaling is possible with vertical scaling i.e. more data storage requires a bigger server. Though multiple RDBMs servers can be added, this is complex and time consuming. | Database scaling is possible with horizontal scaling across multiple servers. |
| SQL Databases follow ACID ((Atomicity, Consistency, Isolation, Durability) properties to ensure that database transactions are reliable. | NoSQL database follows the Brewers CAP theorem (Consistency, Availability and Partition tolerance). |
| SQL Databases get classified as either open source or non-open source. | NoSQL DBs get classified based on the data storage type- Graph DBs, Key-Value store DBs, and Document store DBs. |